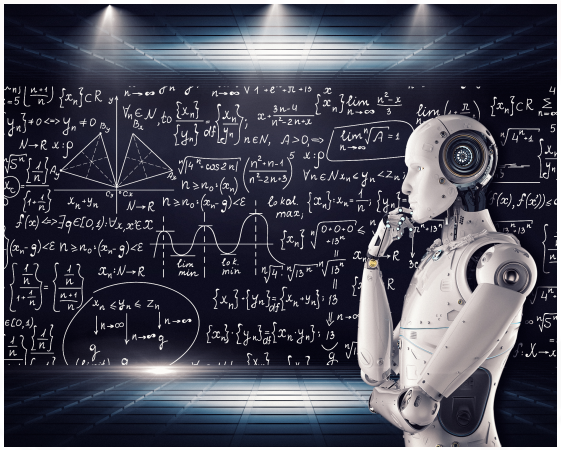
Python Introduction

Dana Golden, Lilia Maliar



Data Science and Machine Learning - **November 30, 2024**

Presentation Outline

What is Python?

- A general-purpose object-oriented high-level programming language
- Valuable for rapid application development and quick scripting
- Quick edit-debug cycle makes high productivity
- An incredibly active programming community

Why Python?

- Much more useful than Stata or R, general purpose

Why Python?

- Much more useful than Stata or R, general purpose
- Python has a massive community of developers working on code, so much code has already been written

Why Python?

- Much more useful than Stata or R, general purpose
- Python has a massive community of developers working on code, so much code has already been written
- Python can do pretty much anything

Why Python?

- Much more useful than Stata or R, general purpose
- Python has a massive community of developers working on code, so much code has already been written
- Python can do pretty much anything
- Extremely low barrier to entry for writing code compared to some languages

Why Python?

- Much more useful than Stata or R, general purpose
- Python has a massive community of developers working on code, so much code has already been written
- Python can do pretty much anything
- Extremely low barrier to entry for writing code compared to some languages
- The default programming language for machine learning

Drawbacks of Python

- Computationally slow, not for scientific computing

Drawbacks of Python

- Computationally slow, not for scientific computing
- Higher barrier to entry than Stata or R, particularly for certain econometric modelling

Drawbacks of Python

- Computationally slow, not for scientific computing
- Higher barrier to entry than Stata or R, particularly for certain econometric modelling
- Not ideal for certain database tasks

Drawbacks of Python

- Computationally slow, not for scientific computing
- Higher barrier to entry than Stata or R, particularly for certain econometric modelling
- Not ideal for certain database tasks
- Can cause problems when used without strong understanding

Types

- Float
- Integer
- String
- Boolean

Data Structures

- Lists: [1,2,'hello',2,1,'fish',1.5]
- Tuples: (1,2,3),('hi','why')
- Dictionaries: {'Dana': 'TA','Lilia':'Professor'}
- Sets: {5,6,4,3}

Lists

- Lists are ordered, changagble, and allow duplicate values
- List are indexed, starting at zero
- Add to lists using append, take from lists using remove
- list[1]=first item in list

Dictionaries

- Dictionaries connect individual 'keys' to 'values'
- Allows you to link and create mapping tables
- Useful for creating trees and more complex data structures
- Dict[key]=value

Variable Assignment

- Variables are assigned in Python using one equals sign
- Equality is checked using two equal signs
- x=True: print(x): True
- False==True: False

Operators

- x=100: y=200: x+y: 300
- x-y: -100
- x/y: 0.5
- x*y: 20000
- x¿=y: False
- x¡=y: True

Math Functions

- math.ceil: Gets the next highest integer, rounds up
- math.floor: Rounds down to the next lowest integer
- math.factorial: Finds the factorial of a value
- math.log: Takes the natural log
- math.exp: Takes the exponent
- random.randint: Generates a random integer between two numbers

If-statements

- If statements allow you to perform operations only if certain conditions are met
- Useful for handling edge cases
- if x¡200:
- print('hi')
- hi

For-loops

- For-loops allow you to iterate over a set of objects or perform an action a certain number of times
- for i in range(10):
- for i in list:

While loops

- While loops perform a task until a certain condition is met
- while x¡200:
- x=x+10
- print(x)

Functions

- Functions start with def(arguments):
- Functions can have keyword arguments and positional arguments
- Functions allow you to reference different tasks that you do routinely or build more complex routines

Installing packages in Python

- When installing Python, add Python to path
- Use pip install ‘package’ to install packages from the command line
- In colab, use !pip install ‘package’
- To use package, import the package

What is Pandas?

- Unlike Stata, as many dataframes as you want can be held in memory
- Pandas takes the best parts of R and integrates them into Python
- Pandas is a way to work with dataframes in Python
- Pandas allows you to do anything you can do in excel or Stata, but faster and more programatically

Summarizing Data Pandas

- df.head(n): Finds first n rows in dataframe
- df.describe(): Provides basic descriptive statistics for columns
- sum(): Sums dataframe
- min(): Finds smallest value in dataset
- max(): Finds largest value in dataset
- count(): counts non-null items in dataset
- median(): Finds data median
- mean(): Finds data mean
- var(): Finds data variance
- std(): Finds data standard deviation
- apply(function): Applies function to each object in dataframe

Grouping Data

- df.groupby(by="col").sum()
- df.groupby(level="ind")

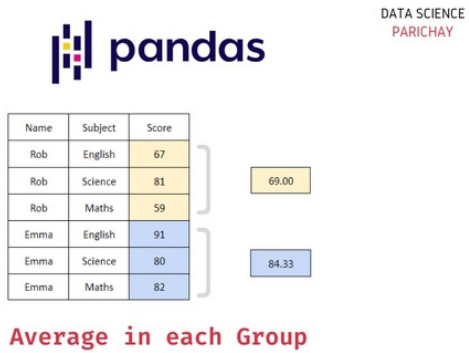


Figure 1: Group by in Pandas

Handling Missing Data

- df.dropna(): Tosses out missing values
- df.fillna(value): replaces missing values with chosen value

Wide to Long

- Melt takes data in wide format and translates to long

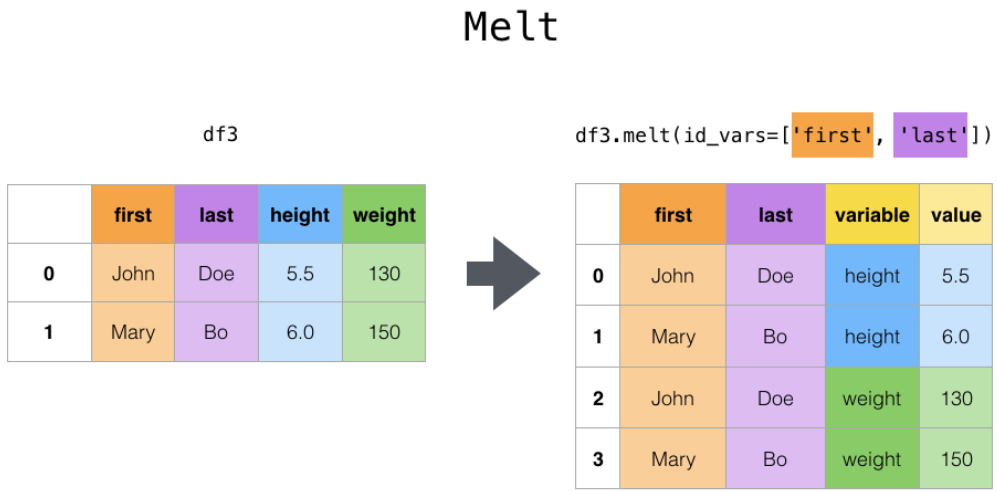


Figure 2: Pandas melt

Long to Wide

- Pivot takes data from long format and puts in wide

df

Combining Datasets: Merge

- Merge adds rows and columns between datasets of common ID
- pd.merge(ydf, zdf, how='outer')

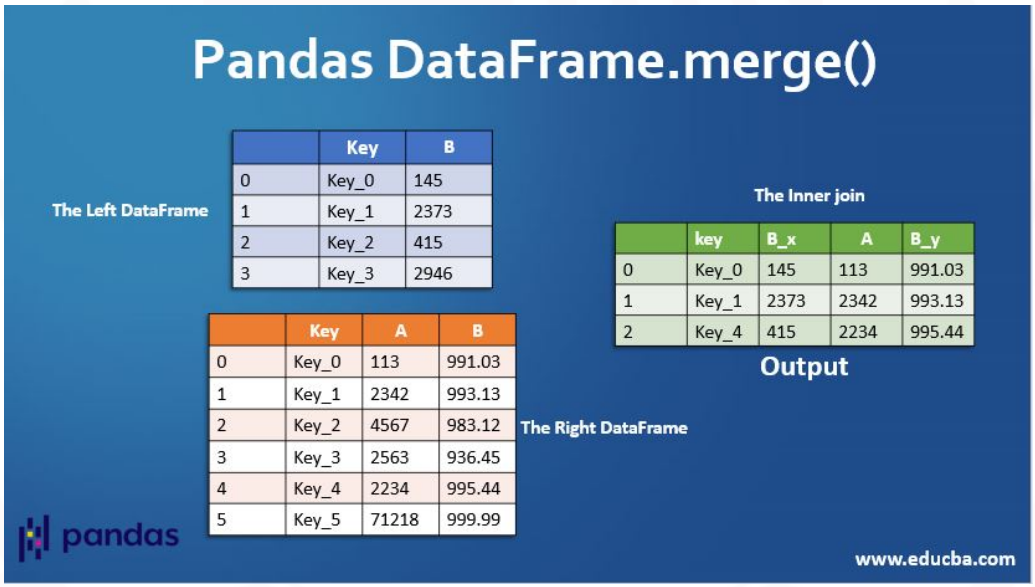
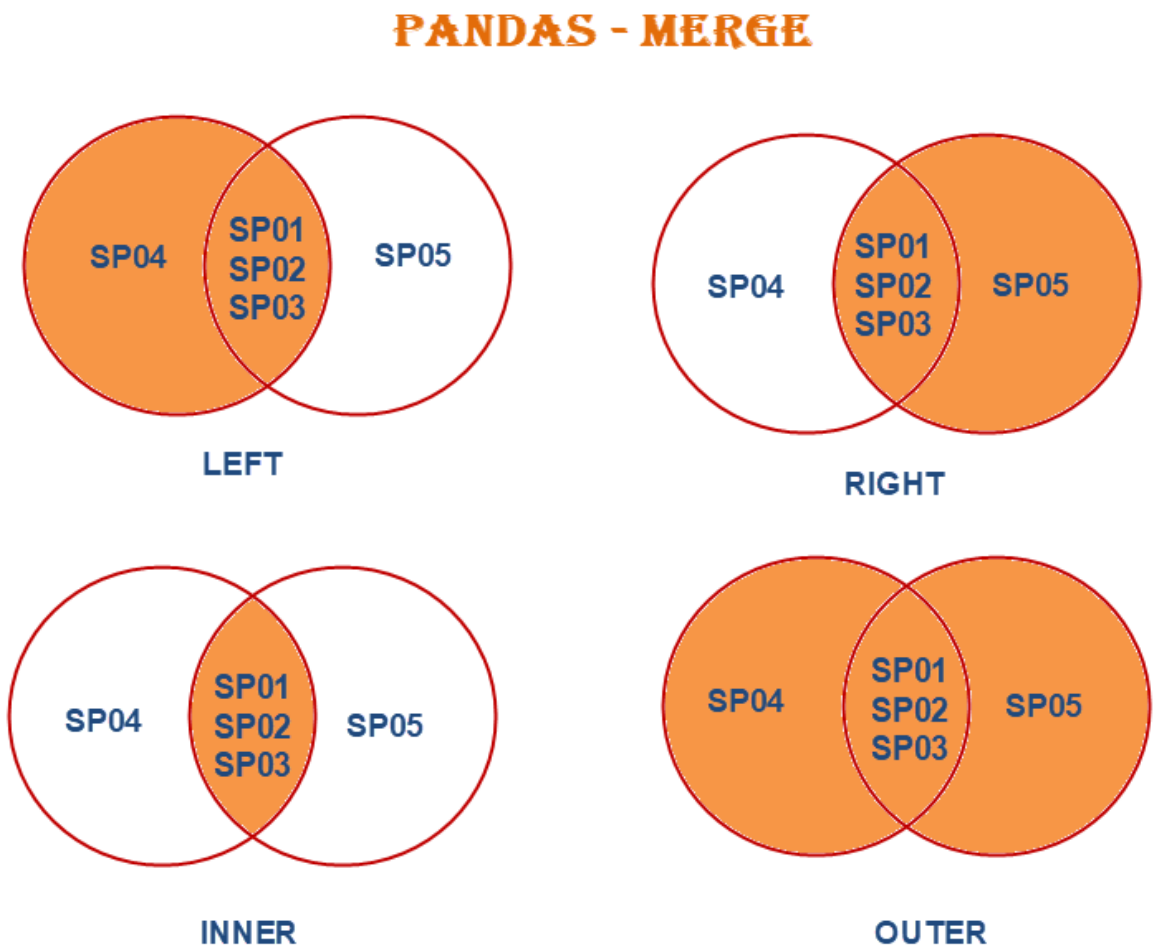


Figure 4: Pandas Merge

Types of Merge



Combine Datasets: Append and Concat

- Append and concat add rows to a pre-existing dataset or extend dataset with new columns
- pd.concat([s1, s2])
- df.append(df2, ignore_index=True)

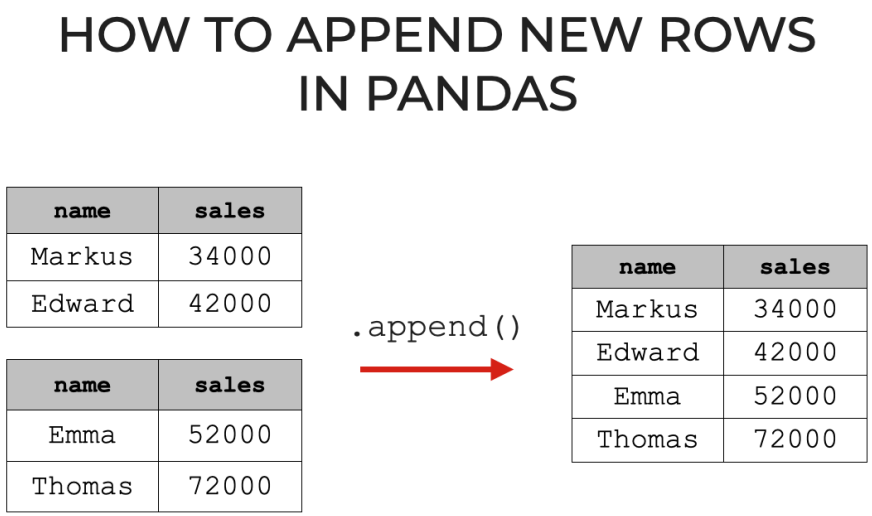


Figure 5: Pandas append

Plotting Pandas

- df.plot.hist(): provides a histogram of all columns
- df.plot.scatter(x='w',y='h'): provides a scatterplot of specific columns

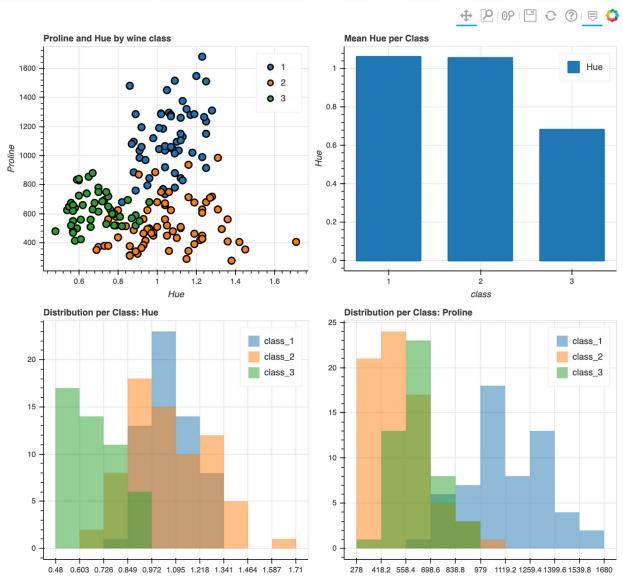


Figure 6: Plotting in Pandas

Window Functions

- df.expanding(): Return an Expanding object allowing summary functions to be applied cumulatively.
- df.rolling(n) Return a Rolling object allowing summary functions to be applied to windows of length n

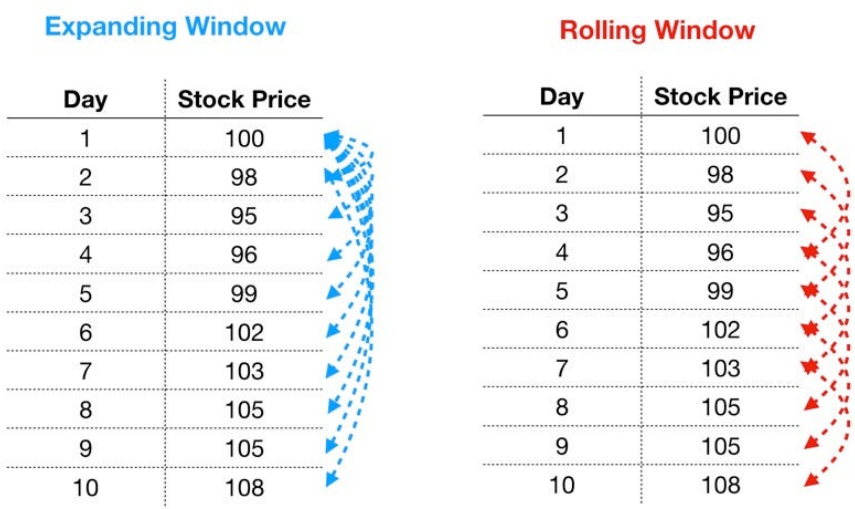


Figure 7: Rolling vs. Expanding Window

What is Numpy?

- Numpy allows you to work with matrices and large datasets in Python
- Numpy dramatically speeds up Python operations: Vectorization
- Numpy is effectively Matlab in Python
- Extremely valuable for econometrics and machine learning

Arrays

- Arrays are the unit of observation in Numpy
- By creating arrays, we can perform numpy operations

Creating Arrays

- np.zeros: Creates a matrix of specified size of zeros
- np.ones: Creates a matrix of specified size of ones
- np.linspace: Returns evenly space numbers over interval, uses count
- np.arange: Same as linspace but uses stepsize
- np.eye: Creates an identity matrix of set size
- np.random.random:

Getting Shape of Array

- array.shape: get dimmensions as tuple
- len(array): get length of array

Array Mathematics

- a = np.array([1,2,3])
- b=np.array([(1.5,2,3), (4,5,6)], dtype = float)
- g=a-b: array([[-0.5,0. ,0.],[-3. , 3. , 3.]])
- b+a: array([[2.5, 4. , 6.],[5. ,7. ,9.]])
- a/b: array([[0.66666667,1. ,1.],[0.25 ,0.4 ,0.5]])
- a*b: array([[1.5, 4. ,9.],[4. , 10. , 18.]])

Other Matrix Math

- np.exp: Finds the power of e of every matrix element
- np.log: Gets natural log of elements in an array
- np.sqrt: Takes square root
- np.sin: Gets sine of each element in erray
- array1.dot(array2): Gets dot product

Basic Matrix Operations

- @: multiply two matrices
- *: elementwise multiply matrices
- .T: Take the transpose
- linalg.eig: computes eignvalues and eigenvectors of array
- linalg.inv: Computes inverse of matrix
- linalg.norm: Computes norm of vector or matrix

Boolean Masking

- Boolean masking allows you to pull from numpy array based on conditions

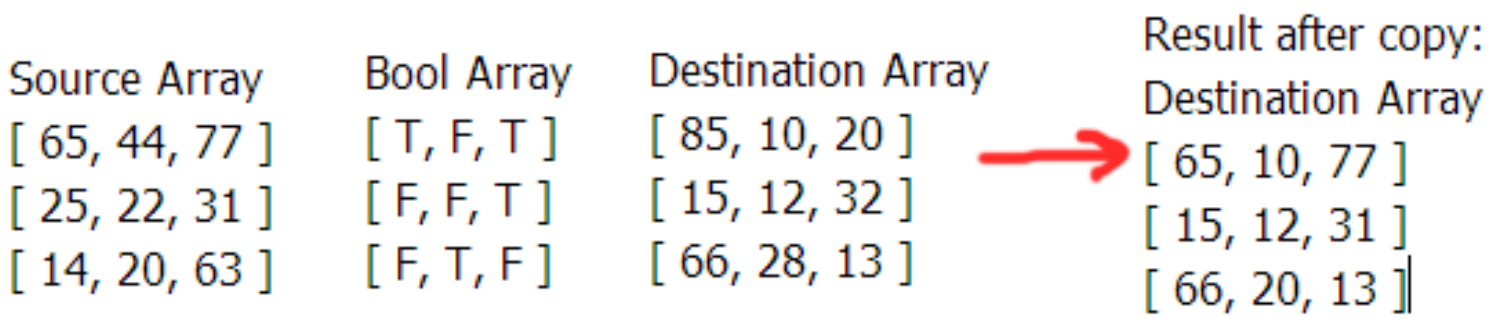


Figure 8: Boolean Masking

Linear Regression using Matrices

- Formula for linear regression: $\beta = (X'X)^{-1}X'y$
- np.linalg.inv(X.T@X)@X.T@y
- np.linalg.lstsq(x,y)

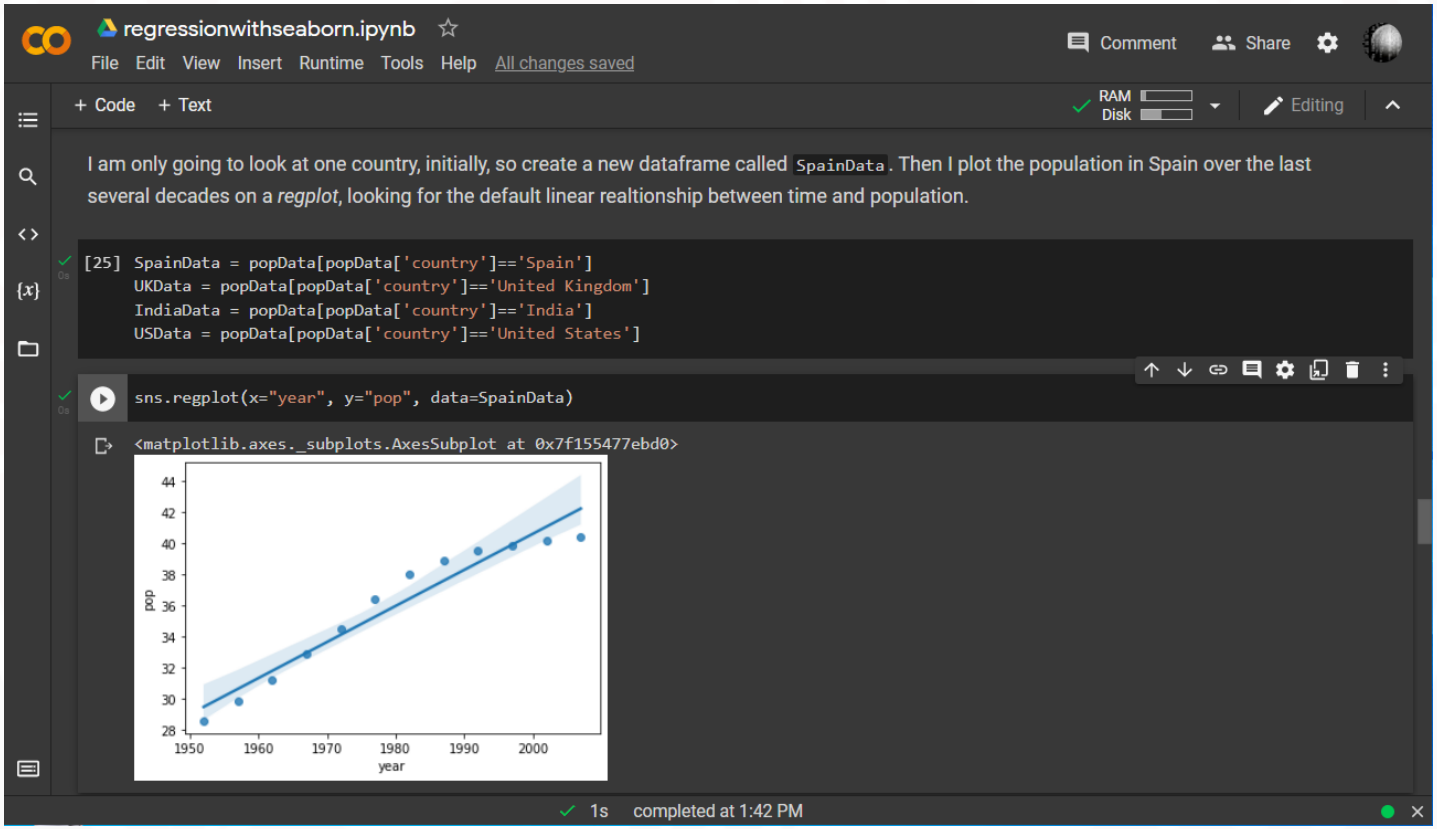
What is a Jupyter Notebook?

- Jupyter notebook allows you to run code in cells rather than in a terminal
- Makes it easier to write and run code
- Useful for sharing code and also creating readable code

How to run a Jupyter Notebook: Command Line

- Go to folder you want to access using cd
- Type: jupyter notebook

How to run a Jupyter Notebook: Colab



Features of Jupyter Notebooks

- You can add text or documentation, even Latex in the notebook
- To run a cell, press cntrl+enter
- can stop code by clicking box in corner
- If things are not working, try restarting the Kernel

What is Matplotlib?

- Matplotlib is Python's default plottling library
- Creates very simple plots but very powerful for basic plotting

Creating Scatter Plot

- fig, ax = plt.subplots()
- for color in ['tab:blue','tab:orange', 'tab:green']:
  - n = 750
  - x, y = np.random.rand(2, n) scale = 200.0 * np.random.rand(n)
  - ax.scatter(x, y, c=color, s=scale, label=color,alpha=0.3, edgecolors='none')
- ax.legend()
- ax.grid(True)
- plt.show()

Scatterplot Example

Scatter Plot using Matplotlib in Python

```
import matplotlib.pyplot as pyplot
# Create data
riding = ((17, 18, 21, 22, 19, 21, 25, 22, 25, 24),(3, 6, 3.5, 4, 5, 6.3, 4.5, 5, 4.5, 4))
swimming = ((17, 18, 20, 19, 22, 21, 23, 19, 21, 24),(8, 9, 7, 10, 7.5, 9, 8, 7, 8.5, 9))
sailing = ((31, 28, 29, 36, 27, 32, 34, 35, 33, 39),(4, 6.3, 6, 3, 5, 7.5, 2, 5, 7, 4))
# Plot the data
pyplot.scatter(x=riding[0], y=riding[1], c='red', marker='s',label='riding')
pyplot.scatter(x=swimming[0], y=swimming[1], c='green',marker='o', label='swimming')
pyplot.scatter(x=sailing[0], y=sailing[1], c='blue',marker='*', label='sailing')
# Configure graph
pyplot.xlabel('Age')
pyplot.ylabel('Hours')
pyplot.title('Activities Scatter Graph')
pyplot.legend()
pyplot.show()
#clcoding.com
```

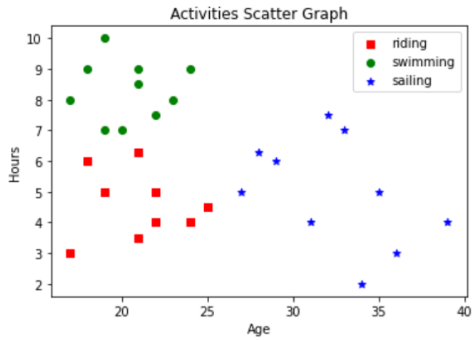


Figure 9: Matplotlib Scatterplot

Creating Bar Chart

- plt.bar(courses, values, color ='maroon', width = 0.4)
- plt.xlabel("Courses offered")
- plt.ylabel("No. of students enrolled")
- plt.title("Students enrolled in different courses")
- plt.show()

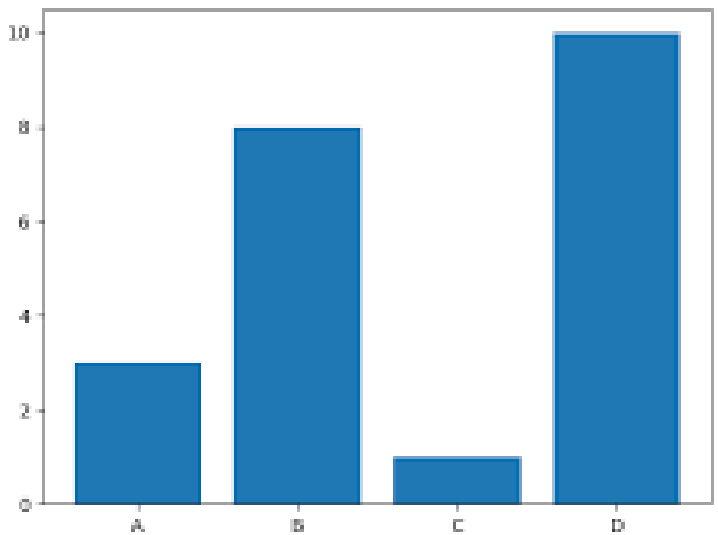


Figure 10: Bar Chart

Various types of Visualizations

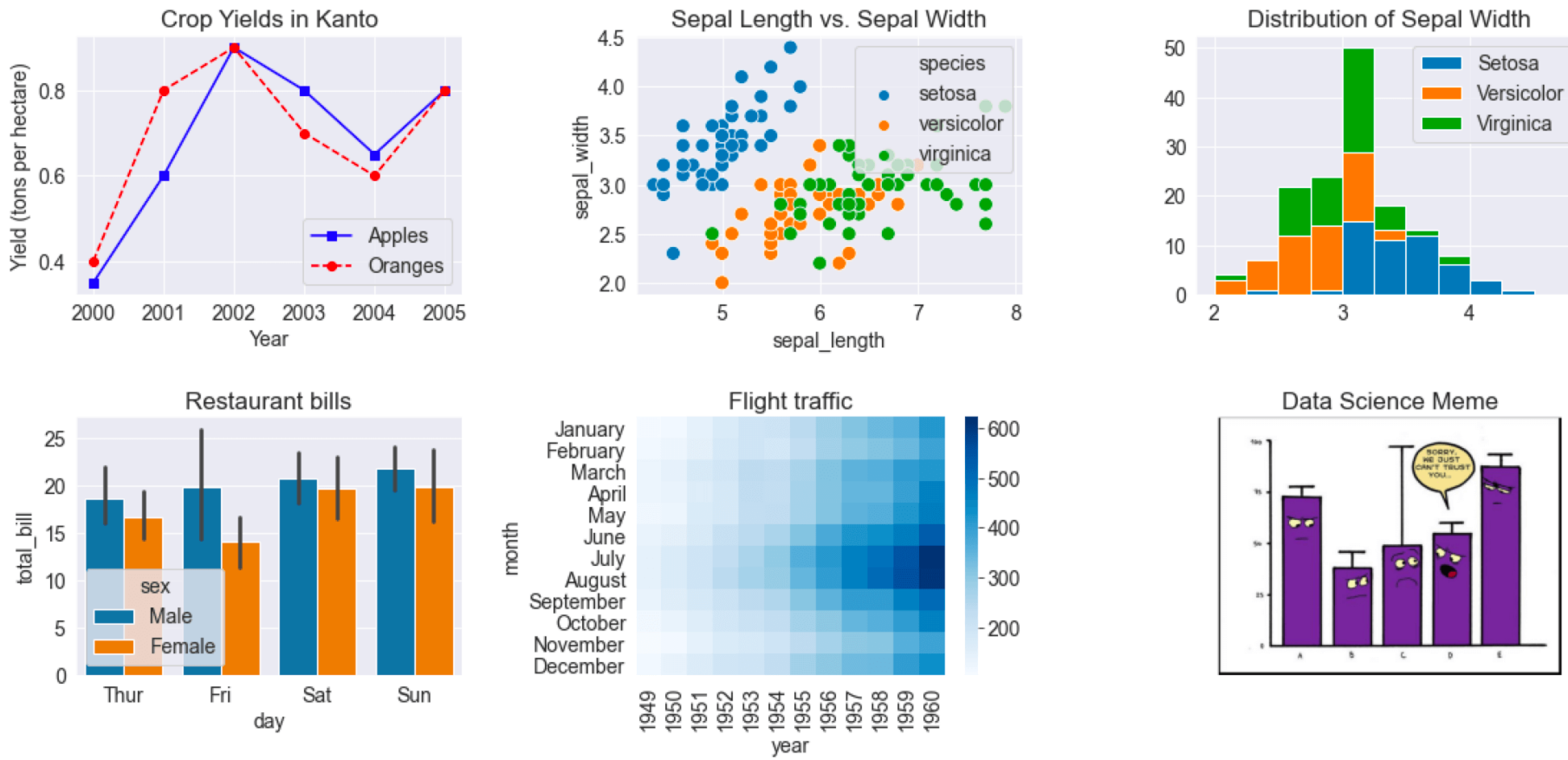


Figure 11: Different Figures in Matplotlib

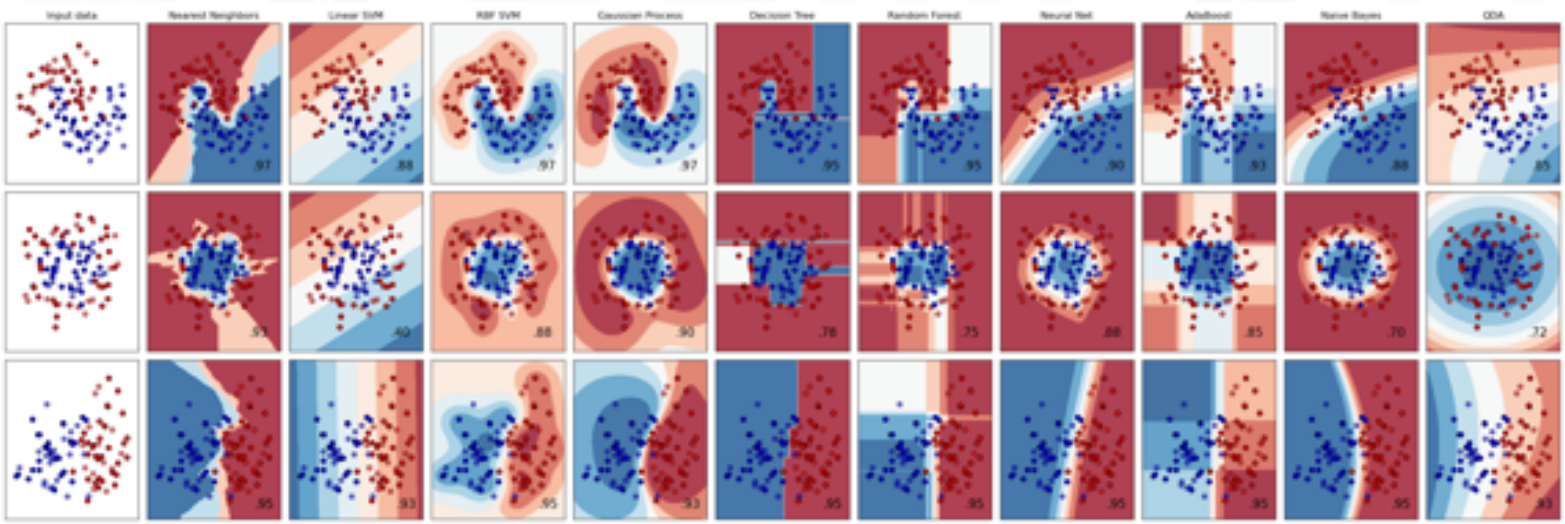
What about dynamic visualizations?

- Matplotlib struggles with dynamic visualizations
- Plotly is an easy to use framework for dynamic visualizations

Figure 12: Plotly Figures

What is Sklearn?

- Sklearn is a general-purpose machine learning library
- Sklearn allows you to use machine learning frameworks right out of the box without additional coding and integrate them into code



How to use Sklearn for machine learning

- Create data: X, y
- Define model: lr =LinearRegression(normalize=True)
- Fit model: lr.fit(X,y)
- Use model for prediction: lr.predict(X,y)
- Determine accuracy of model: accuracy_score(y_test,y_pred)

Sklearn Common Functions

- X_train, X_test, y_train, y_test = train_test_split(X,y,random_state=0)
- lr =LinearRegression(normalize=True)
- lr.fit(X, y)
- knn =neighbors.KNeighborsClassifier(n_neighbors=5)
- knn.fit(X_train,y_train)

What are APIs?

- APIs are how the internet talks to itself

What are APIs?

- APIs are how the internet talks to itself
- APIs speed up the process of getting data

Why use APIs?

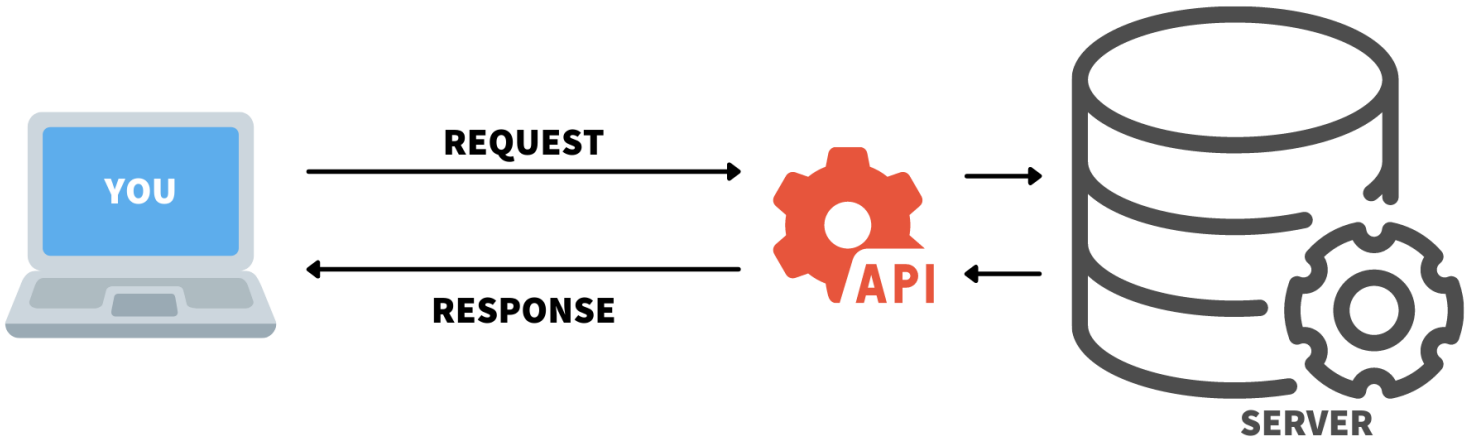
- APIs automate the process of getting data

Why use APIs?

- APIs automate the process of getting data
- APIs allow you to automatically get lots of data or repeatedly get the same data easily

Why use APIs?

- APIs automate the process of getting data
- APIs allow you to automatically get lots of data or repeatedly get the same data easily
- Useful for building apps and developing data pipelines



Requests in Python

- Requests is Python's framework for accessing websites and querying APIs
- Get method: Gets content from a URL
- Post method: requests server accept data in request
- Arguments: URL, params kwargs

API calls example

```python
# IMPORT
import json
import os
import requests

# INITIALIZE
apiToken = os.environ['NETBOX_APITOKEN']
nbApiHeaders = {'Authorization': 'Token ' + apiToken}

nbBaseURL = 'http://172.22.45.1/api'
nbApiUrl = '/dcim/regions/'
nbApiQuestion = nbBaseURL + nbApiUrl

# MAIN
# Retrieve JSON blob from NetBox
nbApiRawAnswer = requests.get(nbApiQuestion, headers=nbApiHeaders)
# Prettyprint the JSON blob
print(json.dumps(nbApiRawAnswer.json(), indent=4))
```

Thank You So Much!

Resources

- Data Camp
- Code Academy
- LeetCode
- Econometrics in Python
- Python Tutorials